Exercise set 5:

8.1 Explain the difference between internal and external fragmentation.

8.2 Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory binding tasks of the linkage editor?

8.3 Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?

9.8 Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

- LRU replacement
- FIFO replacement
- Optimal replacement

9.12 Discuss situations in which the MFU page-replacement algorithm generates fewer page faults than the LRU page-replacement algorithm. Also discuss under what circumstances the opposite holds.

9.25 Consider a system that allocates pages of different sizes to its processes. What are the advantages of such a paging scheme? What modifications to the virtual memory system provide this functionality?

Extra:

Consider mapping a virtual memory of 1GB onto a physical memory organized into 256 page frames of 4KB each. Moreover, assume that the smallest addressable unit is 1 byte.

a) Does the page table fit in the main memory? Motivate your answer with a calculation.
b) Does the frame table fit in a single page? Motivate your answer with a calculation.

Given the same physical memory organization as above and a memory management strategy that always keeps at least 1 page frame per active process resident in main memory for paging purposes:

c) What is the maximum size of the virtual address space given that any memory access may yield at most two page faults?
d) Give the corresponding address map (algorithm)

Questions:

Q1)

Consider a system with 640KB of main memory. The following blocks are to be requested and released:

block	A	B	C	D	E	F
size (KB)	200	200	150	80	80	240

Obviously, not all blocks may be accommodated at the same time. We say an allocation failure occurs when there is no hole large enough to satisfy a given request. Devise a sequence of requests and releases that results in an allocation failure under:

(a) the first fit policy, but not the best fit policy

(b) the best fit policy, but not the first fit policy

For each of the two situations, draw a diagram that shows what parts of memory are allocated at the time of failure. (Hint: there is no systematic approach to this problem; you simply need to experiment with different sequences.)

Q2)

Consider a system with 4.2 MB of main memory using variable partitions. At some point in time, the memory will be occupied by three blocks of code/data as follows:

starting address	length
1,000,000	1 MB
2,900,000	.5 MB
3,400,000	.8 MB

The system uses the best-fit algorithm. Whenever a request fails, memory is compacted using one of the following schemes:

a. **All blocks are shifted towards address zero (see Figure 1b).**
b. **Starting with the lowest address block, blocks are shifted towards address zero until a hole big enough for the current request is created (see Figure 1c).**

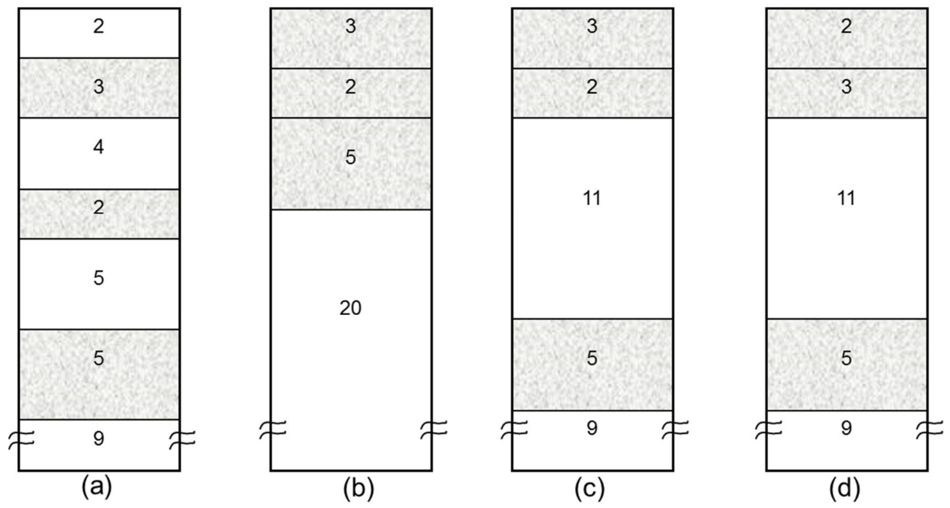


Figure 1

Assume that three new blocks with the respective sizes 0.5 MB, 1.2 MB, and 0.2 MB are to be loaded (in the given sequence).

(a) Show the memory contents after all three requests have been satisfied under the two different memory compaction schemes.

(b) How many bytes of memory had to be copied in each case?

Q3)

In a system with paging and segmentation, each virtual address (s, p, w) requires 3 memory accesses. To speed up the address translation, a translation look-aside buffer holds the components (s, p) together with the corresponding frame number. If each memory access takes m ns and the access to the translation look-aside buffer takes $m/10$ ns, determine the hit ratio necessary to reduce the average access time to memory by 50%.

Q4)

Assuming a physical memory of four page frames, give the number of page faults for the reference string *abgadeabadegde* for each of the following policies. (Initially, all frames are empty.)

(a) MIN

(b) FIFO

(c) Second Chance algorithm

(d) Third Chance algorithm (assume that all accesses to page b are write requests)

(e) LRU

(f) VMIN with $\tau = 2$

(g) WS with $\tau = 2$

Exercise set 5:

8.1 Explain the difference between internal and external fragmentation.

Answer:

a. Internal fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released.

b. External fragmentation is unused space between allocated regions of memory. Typically external fragmentation results in memory regions that are too small to satisfy a memory request, but if we were to combine all the regions of external fragmentation, we would have enough memory to satisfy a memory request.

8.2 Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory binding tasks of the linkage editor?

Answer:

The linkage editor has to replace unresolved symbolic addresses with the actual addresses associated with the variables in the final program binary. In order to perform this, the modules should keep track of instructions that refer to unresolved symbols. During linking, each module is assigned a sequence of addresses in the overall program binary and when this has been performed, unresolved references to symbols exported by this binary could be patched in other modules since every other module would contain the list of instructions that need to be patched.

8.3 Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?

Answer:

a. First-fit:
b. 212K is put in 500K partition
c. 417K is put in 600K partition
d. 112K is put in 288K partition (new partition 288K = 500K – 212K)
e. 426K must wait
f. Best-fit:
g. 212K is put in 300K partition
h. 417K is put in 500K partition
i. 112K is put in 200K partition
j. 426K is put in 600K partition
k. Worst-fit:
l. 212K is put in 600K partition
m. 417K is put in 500K partition

n. 112K is put in 388K partition
o. 426K must wait

In this example, best-fit turns out to be the best.

8.12

Answer:

a. page = 3, offset = 13

b. page = 41, offset = 11

c. page = 210, offset = 161

d. page = 634, offset = 784

e. page = 1953, offset = 129

9.8 Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

- LRU replacement
- FIFO replacement
- Optimal replacement

Answer:

- 18
- 17
- 13

9.12 Discuss situations in which the MFU page-replacement algorithm generates fewer page faults than the LRU page-replacement algorithm. Also discuss under what circumstances the opposite holds.
Answer:
Consider the sequence in a system that holds four pages in memory: 1 2 3 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence "1 2 3 4 4 4 5 1," the LRU algorithm makes the right decision.

9.25 Consider a system that allocates pages of different sizes to its processes. What are the advantages of such a paging scheme? What modifications to the virtual memory system provide this functionality?

Answer:

The program could have a large code segment or use large-sized arrays as data. These portions of the program could be allocated to larger pages, thereby decreasing the memory overheads associated with a page table. The virtual memory system would then have to maintain multiple free lists of pages for the different sizes and also needs to have more complex code for address translation to take into account different page sizes.

A1)

(a) Sequence: req A, req C, req B, req D, rel C, rel D, req E, rel A, req F – fail

Memory is allocated as follows:
200 KB: unallocated
80 KB: block E
70 KB: unallocated
200 KB: block B
70 KB: unallocated

(b) Sequence: req C, req A, req B, req D, rel C, rel D, req E, rel B, req F – fail

Memory is allocated as follows:
150 KB: unallocated
200 KB: block A
200 KB: unallocated
80 KB: block E
10 KB: unallocated

A2)

(a) Let's refer to the three original blocks as X, Y, Z, and the three new blocks as A, B, C. Then memory is occupied as follows:
Memory under policy 1:

block name	starting address
A	0
X	500,000
Y	1,500,000
Z	2,000,000
B	2,800,000
C	4,000,000

Memory under policy 2:

block name	starting address
A	0
X	500,000
B	1,500,000
C	2,700,000
Y	2,900,000
Z	3,400,000

(b) Under policy 1, all the three old blocks had to be moved, resulting in 2.3 MB of data copied.
Under policy 2, only block X had to be moved, resulting in 1 MB of data copied.

A3)

Without the TLB, each virtual memory translation needs 3 memory accesses, or $3m$ ns. With the TLB, some fraction of virtual memory accesses will still need $3m$ while others will only need $m + m/10$), i.e., one memory access plus the time to access the TLB. If h is the hit ratio, then, on average, a virtual memory translation will require $h * (m + m/10) + (1 - h) * 3m$. To reduce the original access time of $3m$ by 50%, i.e., to $3m/2$, we need to equate the above expression with $3m/2$:
$h * (m + m/10) + (1 - h) * 3m = 3m/2$
This yields $h = 0.79$, i.e., about 80% of all virtual memory references must find a match in the TLB.

A4)

(a) MIN - 6 page faults

```
 1   2   3   4   5   6   7   8   9  10  11  12  13  14
-----------------------------------------------------
 a   a   a   a   a   a   a   a   a   a   a   g   g   g
 -   b   b   b   b   b   b   b   b   b   b   b   b   b
 -   -   g   g   g   e   e   e   e   e   e   e   e   e
 -   -   -   -   d   d   d   d   d   d   d   d   d   d
-----------------------------------------------------
 *   *   *       *   *                       *
```

(b) FIFO - 10 page faults

```
 1   2   3   4   5   6   7   8   9  10  11  12  13  14
-----------------------------------------------------
>a  >a  >a  >a  >a   e   e   e   e   e   e  >e   d   d
 -   b   b   b   b  >b   a   a   a   a   a   a  >a   e
 -   -   g   g   g   g  >g   b   b   b   b   b   b  >b
 -   -   -   -   d   d   d  >d  >d  >d  >d   g   g   g
-----------------------------------------------------
 *   *   *       *   *   *   *               *   *   *
```

(c) second chance algorithm - 10 page faults

1	2	3	4	5	6	7	8	9	10	11	12	13	14
a/1	a/1	a/1	a/1	>a/1	e/1	e/1	e/1	e/1	e/1	e/1	>e/0	d/1	d/1
-	b/1	b/1	b/1	b/1	>b/0	a/1	a/1	a/1	a/1	a/1	a/0	>a/0	e/1
-	-	g/1	g/1	g/1	g/0	>g/0	b/1	b/1	b/1	b/1	b/0	b/0	>b/0
-	-	-	-	d/1	d/0	d/0	>d/0	>d/0	>d/1	>d/1	g/1	g/1	g/1
*	*	*		*	*	*	*				*	*	*

(d) third chance algorithm - 9 page faults

1	2	3	4	5	6	7	8	9	10	11	12	13	14
a10	a10	a10	a10	>a10	e10	e10	e10	e10	e10	e10	>e00	d10	d10
-	b11	b11	b11	b11	>b01	b00	b11	b11	b11	b11	b01	>b01	b00
-	-	g10	g10	g10	g00	a10	a10	a10	a10	a10	a00	a00	e10
-	-	-	-	d10	d00	>d00	>d00	>d00	>d10	>d10	g10	g10	>g10
*	*	*		*	*	*					*	*	*

(e) LRU - 7 page faults

1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	a	a	a	a	a	a	a	a	a	a	a	a	a
-	b	b	b	b	e	e	e	e	e	e	e	e	e
-	-	g	g	g	g	g	b	b	b	b	g	g	g
-	-	-	-	d	d	d	d	d	d	d	d	d	d
a	b	g	a	d	e	a	b	a	d	e	g	d	e
	a	b	c	a	d	e	a	b	a	d	e	g	d
		a	b	g	a	d	e	e	b	a	d	e	g
				b	g	g	d	d	e	b	a	a	a
*	*	*		*	*		*				*		

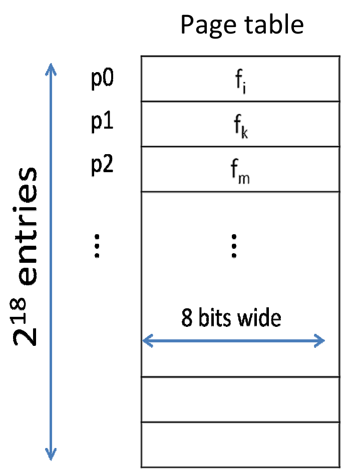
(f) VMIN (with $\tau = 2$) - 9 page faults

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	v	v	v	v	v	v	v	v	v					
b		v						v						
d					v					v	v	v	v	
e						v					v	v	v	v
g			v									v		
	*	*	*		*	*		*		*	*	*		

Question 2: We have the following data

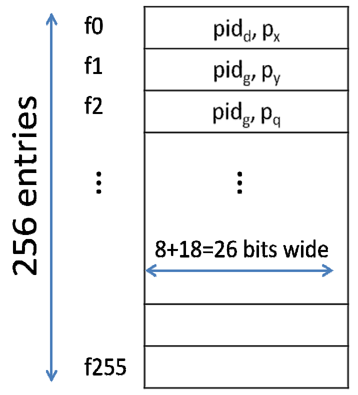
(1) Virtual mem. size: 1GB = 8Gbits
(2) Page size: 4KB = 32 Kbits (= frame size)
(3) #Frames = 256 = 2^8, which means we need 8 bits to represent a frame
(4) Physical memory: 256 x 4KB = 1MB = 8Mbits
(5) Smallest addressable unit: 1B
(6) From (2) and (5) → #bits to address a byte (used as offset): 12bits

a) Max # pages is 1GB/4KB = 256K → 2^{18}



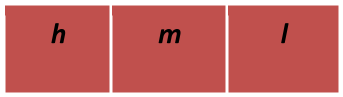
Then the size of the page table is 2^{18} B = 256KB, which fits in the 1 MB physical memory.

b) Frame table consists of 256 page addresses, where each page is addressable using 26 bits (assuming 256 processes: 8 bits for addressing a process, 18 bits for addressing a page within a process).



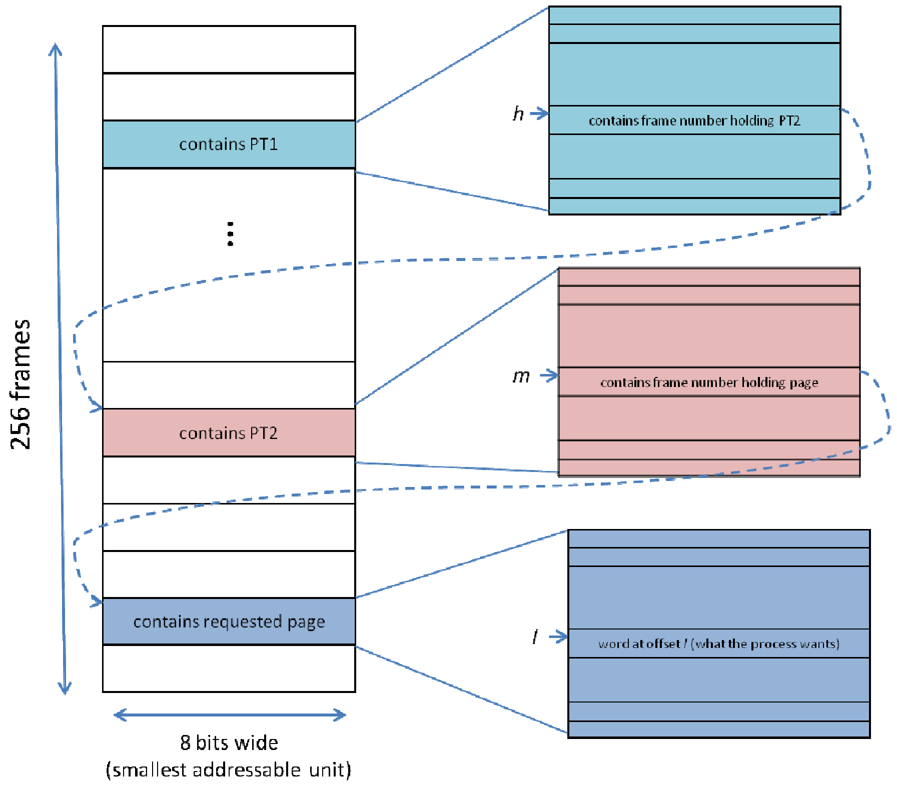
Then the size of the frame table is 256 x 26 bits = 6656 bits, which is much less than the frame size (32 Kbits). Note that 32K = 256 x 128 → the frame table entries can be at most 128 bits wide. This means that pid bits can consist at most of 110 bits (overkill).

c) 1 page frame per active process keeps the page table. If multiple levels of page tables exist, then this frame keeps the first level. Hence, there are no page faults for the first level. If we are allowed max 2 page faults, then there must be 2 levels of page tables, namely PT1 and PT2 (1 page fault for bringing the page-table level 2 into the memory, 1 page fault for bringing the required page into the memory). Then, a virtual address looks like this:



where *h*, *m* and *l* denote the higher order, mid order and lower order bits in the virtual address (12 bits each). Note that PT1 is always in the reserved physical memory frame.

- *h* is the index (in PT1) to the entry for PT2.
- This entry contains a *pointer* to the beginning of the frame that contains PT2.
- *m* is the index (in PT2) to the entry for the page requested.
- This entry contains a *pointer* to the beginning of the frame that contains requested page.
- *l* is the index (the requested page) to the requested instruction (the offset).



256 frames
contains PT1
contains PT2
contains requested page
8 bits wide
(smallest addressable unit)
h
contains frame number holding PT2
m
contains frame number holding page
l
word at offset *l* (what the process wants)

d) *pa = *(*(PTR + h) + m) + l*

where PTR is a register value that points to the beginning of the page table level PT1.